例子

A.在生日相同一题中，按照生日的大小从小到大排序，并且生日相同的人还要保证稳定性：

struct in

{

char name[22];

int month;

int date;

int index;//index=100\*month+date

int no;//no是排序稳定性的关键，no等于alex的角标（保证了生日相同的人no不同）

int party;

}alex[105];

bool cmp(const in&a,const in&b)//第一，写的cmp是bool类型；第二，类型是in不是alex；第三，加&；第四，类型之后是&，然后是名字；第五，加const比较安全

{

if (a.index != b.index)

return((a.index)<(b.index));//这里都是默认从小到大排序的，如果有问题，删改符号

else

return((a.no)<(b.no));//这里达到稳定性排序的目的

}

sort(alex, alex+n,cmp);//alex是头地址，直接alex+n即可

B.最小字典序：将相邻两个字符串直接拼接后比较

bool cmp(string& a,string& b)

{

return a + b < b + a;//注意到你的string是可以直接相加的，就是一个拼接函数

}

C.好学的明明：注意到在qsort里面，无法排序带string的结构体，但是使用sort是可以的

同时，对于string有很多操作对于char其实很复杂；

譬如string可以直接赋值: out[j].type = "0";

同样的功能在char里需要strcoy(out[j].type，"0");

其次，string可以直接判断等于：if (out[j].type == "0")

同样的功能在char里面需要用if(strcmp(out[j].type,"0")==0)

注意到，strcmp在前者字典序大的时候是+1，后者字典序大的时候是-1；在相同时是-1；所以判断不等最好直接用if(strcmp(out[j].type,"0")!=0)

string还可以直接相加实现字符串的拼接；

另外，经验表明输出字符串的题不要试图先拼接好了再输出(特别是用char的时候)；算好了直接打印输出即可；

关于merge

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<cmath>

#include<cstring>

#include<algorithm>

using namespace std;

struct fuck

{

int value;

int no;

}in[5],out[5],push[14];

bool cmp(const fuck&a,const fuck&b)

{

if (a.value != b.value) { return a.value < b.value; }

else { return a.no < b.no; }

}

int main() {

for (int i = 0; i < 5; i++)

{

cin >> in[i].value;

in[i].no = i;

}

for (int i = 0; i < 5; i++)

{

cin >> out[i].value;

out[i].no = i+10;

}

sort(in, in + 5, cmp);

sort(out, out + 5, cmp);

merge(in, in + 5, out, out + 5, push,cmp);

for (int i = 0; i < 10; i++)

cout << push[i].value << " "<<push[i].no<<endl;

}

merge的功能是将两个已经有序的序列排成一个新的有序序列，格式是merge(in, in + 5, out, out + 5, push,cmp);merge和sort连用可以解决一些奇怪问题

这两天在刷leetcode时看到了一些关于排序的题目，其中遇到各种大神对sort与qsort的使用，个人在使用时对比发现了一些sort与qsort的区别，并对它们的使用方法进行了一些总结介绍。

在平时编程时，排序是一种经常要用到的操作。如果每次都自己临场写)排序算法，不仅容易出错，而且浪费宝贵的时间。在C++的STL里面有两个sort与qsort可以直接用于对各种类型的数据以及容器进行排序。

**1、qsort**

qsort函数定义在头文件<algorithm>中，使用时需要include该头文件

功 能: 使用快速排序例程进行排序

用 法: void qsort(void \*base, int nelem, int width, int (\*fcmp)(const void \*,const void \*)); 　　   
参数：1、待排序数组首地址； 2、数组中待排序元素数量； 3、各元素的占用空间大小； 4、指向函数的指针，用于确定排序的顺序   
比如：对一个长为1000的数组进行排序时，int a[1000]; 那么base应为a，num应为 1000，width应为 sizeof(int)，cmp函数随自己的命名。

使用方法：

void qsort( void \*base, size\_t num, size\_t width, int (\_\_cdecl \*compare )   
int compare (const void \*elem1, const void \*elem2 ) );

qsort（quicksort）主要根据你给的比较条件给一个快速排序，主要是通过指针移动实现排序功能。排序之后的结果仍然放在原来数组中。

参数意义如下:

base:需要排序的目标数组开始地址   
num:目标数组元素个数   
width:目标数组中每一个元素长度   
compare:函数指针，指向比较函数（这个函数是要自己写的，sort中默认升序）

（1）对int类型数组排序   
int num[100];   
int cmp ( const void \*a , const void \*b )   
{   
return \*(int \*)a - \*(int \*)b;   
}

可见：参数列表是两个空指针，现在他要去指向你的数组元素。所以转型为你当前的类型，然后取值。升序排列。

qsort(num,100,sizeof(num[0]),cmp);

（2）对char类型数组排序（同int类型）

 char word[100];

int cmp( const void \*a , const void \*b )

{

return \*(char \*)a - \*(int \*)b;

}

qsort(word,100,sizeof(word[0]),cmp);

（3）对double类型数组排序（特别要注意）

double in[100];

int cmp( const void \*a , const void \*b )   
{   
return \*(double \*)a > \*(double \*)b ? 1 : -1;   
}  
返回值的问题，显然cmp返回的是一个整型，所以避免double返回小数而被丢失。    
qsort(in,100,sizeof(in[0]),cmp)；

（4）对结构体一级排序

struct In   
{   
double data;   
int other;   
}s[100]

//按照data的值从小到大将结构体排序,关于结构体内的排序关键数据data的类型可以很多种，参考上面的例子写   
int cmp( const void \*a ,const void \*b)   
{   
return (\*(In \*)a).data > (\*(In \*)b).data ? 1 : -1;   
}   
qsort(s,100,sizeof(s[0]),cmp);

（5）对结构体二级排序

struct In   
{   
int x;   
int y;   
}s[100];

//按照x从小到大排序，当x相等时按照y从大到小排序   
int cmp( const void \*a , const void \*b )   
{   
struct In \*c = (In \*)a;   
struct In \*d = (In \*)b;   
if(c->x != d->x) return c->x - d->x;   
else return d->y - c->y;   
}   
qsort(s,100,sizeof(s[0]),cmp);

（6）对字符串进行排序   
char str[100][100];   
int cmp(const void\* a,const void\* b )

{

return strcmp((char \*)a,(char\*)b);

}

qsort(str,n,sizeof(str[0]),cmp);

struct In   
{   
int data;  
char str[100];   
}s[100];

//按照结构体中字符串str的字典顺序排序   
int cmp ( const void \*a , const void \*b )   
{   
return strcmp( (\*(In \*)a)->str , (\*(In \*)b)->str );   
}   
qsort(s,100,sizeof(s[0]),cmp);

使用qsort一个比较麻烦的地方在于需要自己编写compare即比较函数，而sort默认为升序排序，在大多数情况下，使用sort会更方便一些。

**2、sort**

sort函数，与qsort同为排序函数，复杂度为n\*log2(n)。sort()定义在头文件<algorithm>中。sort函数是标准模板库的函数，已知开始和结束的地址即可进行排序，可以用于比较任何容器（必须满足随机迭代器），任何元素，任何条件，执行速度一般比qsort要快。另外，sort()是类属函数，可以用于比较任何容器，任何元素，任何条件。   
注意：缺省是升序排序。

sort 使用时得注明：using namespace std; 或直接打 std::sort() 还得加上 #include<algorithm>

例：   
#include<iostream>   
#include<algorithm>   
using namespace std;   
int main()  
{   
int a[20];   
for(int i=0;i<20;++i)   
cin>>a[i];   
sort(a,a+20); //范围，很明显这里是a+20 注意，这是必要的，如果是a+19   
for(i=0;i<20;i++) //最后一个值a[19]就不会参与排序。   
cout<<a[i]<<endl;   
return 0;   
}

sort是qsort的升级版，如果能用sort尽量用sort，使用也比较简单，不像qsort还得自己去写 cmp 函数，只要注明 使用的库函数就可以使用，参数只有两个（如果是普通用法）头指针和尾指针；   
默认sort排序后是升序，如果想让他降序排列，可以使用自己编的cmp函数   
bool compare(int a,int b)   
{   
return a>b; //www.cdtarena.com降序排列，如果改为return a<b，则为升序

}   
sort(\*a,\*b,cmp);

例子：  
#include<iostream>  
#include<algorithm>  
using namespace std;  
bool cmp (const int a, const int b)  
{  
    return a > b;  
}  
int main()  
{  
    int data[5];  
    for(int i = 0; i < 5; i++)  
        cin >> data[i];  
    sort(data, data + 5, cmp);  
    return 0;  
}

sort函数可以传两个参数或三个参数。第一个参数是要排序的区间首地址，第二个参数是区间尾地址的下一地址。也就是说，排序的区间是[a,b)。简单来说，有一个数组int a[100]，要对从a[0]到a[99]的元素进行排序，只要写sort(a,a+100)就行了，默认的排序方式是升序。 排序的数据类型不局限于整数，只要是定义了小于运算的类型都可以，比如字符串类string。如果是没有定义小于运算的数据类型，或者想改变排序的顺序，就要用到第三参数——比较函数。比较函数是一个自己定义的函数，返回值是bool型，它规定了什么样的关系才是“小于”。想把刚才的整数数组按降序排列，可以先定义一个比较函数cmp:   
bool cmp(int a,int b)  
{  
    return a>b;  
}

排序的时候就写sort(a,a+100,cmp);   
假设自己定义了一个结构体node:   
struct node{  
    int a;  
    int b;  
    double c;  
};   
有一个node类型的数组node arr[100]，想对它进行排序：先按a值升序排列，如果a值相同，再按b值降序排列，如果b还相同，就按c降序排列。就可以写这样一个比较函数：   
以下是代码片段：   
bool cmp(node x,node y)  
{  
     if(x.a!=y.a)  return x.a  
     if(x.b!=y.b)  return x.b>y.b;  
     return  return x.c>y.c;  
}

排序时写sort(arr,a+100,cmp);

**3、sort与qsort的对比**

（1）最直观的差别，函数形式不一样，

qsort的使用方式为：

void qsort( void \*base, size\_t num, size\_t width, int (\_\_cdecl \*compare )

sort的使用方式为：

template <class RandomAccessIterator>

void sort ( RandomAccessIterator first, RandomAccessIterator last )；

template <class RandomAccessIterator, class Compare>

void sort ( RandomAccessIterator first, RandomAccessIterator last, Compare comp );

sort有二个参数与三个参数版本，两个参数默认升序排序，第三个参数可用于指定比较函数，调整排序方式。

（2）compare函数的写法也是不一样的。

qsort的compare函数写法为：

int compare (const void \*elem1, const void \*elem2 ) );

sort的compare函数返回的是bool值；

（3）sort是一个改进版的qsort. std::sort函数优于qsort的一些特点：对大数组采取9项取样，更完全的三路划分算法，更细致的对不同数组大小采用不同方法排序。如果能用sort尽量用sort，使用也比较简单，不像qsort还得自己去写 cmp 函数，只要注明 使用的库函数就可以使用，参数只有两个（如果是普通用法）头指针和尾指针；默认sort排序后是升序，如果想让他降序排列，可以使用自己编的cmp函数